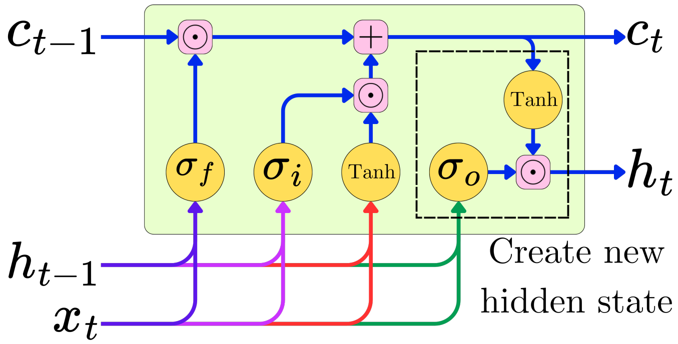
Understanding The LSTM Layer

Damien Benveniste

The AiEdge



LSTM

The Long Short-Term Memory (LSTM), introduced by Sepp Hochreiter and Jürgen Schmidhuber[1] in 1997, was specifically designed to avoid those vanishing and exploding gradients, primarily through a new gated architecture. The LSTM layer had a transformative impact on Natural Language Processing. Historically speaking, LSTMs represented a critical bridge between simple RNNs and modern Transformer-based architectures, demonstrating that neural networks could effectively process sequential data with long-range dependencies.

1 The Architecture

There are three inputs to the LSTM cell: the input $\mathbf{x}_t$ from the data at the time step t, the hidden state $\mathbf{h}_{t-1}$ of the previous iteration, and a new parameter $\mathbf{c}_{t-1}$ called the "cell state." Both $\mathbf{h}_{t-1}$ and $\mathbf{c}_{t-1}$ act as memory for the cell. We combine the information of the current input and the previous hidden state with three mixing "gates":

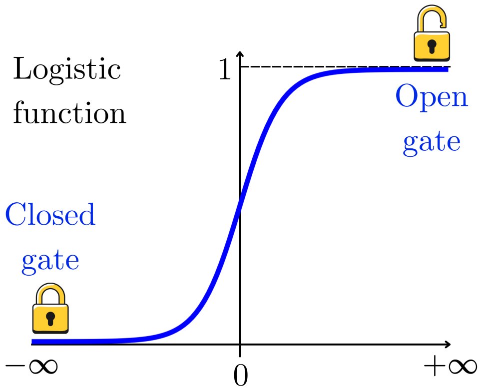
- The forget gate:
$$\mathbf{f}_t = \sigma\left(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + \mathbf{b}_f\right) \tag{1}$$
- The input gate:
$$\mathbf{i}_t = \sigma\left(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + \mathbf{b}_i\right) \tag{2}$$
- The output gate:
$$\mathbf{o}_t = \sigma\left(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + \mathbf{b}_o\right) \tag{3}$$

$W_\circ$ and $U_\circ$ are transition matrices (linear layers) and $\mathbf{b}_\circ$ are the bias vectors. Here, $\sigma(\cdot)$ is the logistic sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

The logistic function ranges from 0 to 1, and we call those "gates" because they allow us to regulate the flow of information passing through. Let's say we have a quantity z. If we multiply it by $\sigma(x)$, we can tune on and off the resulting quantity depending on x:

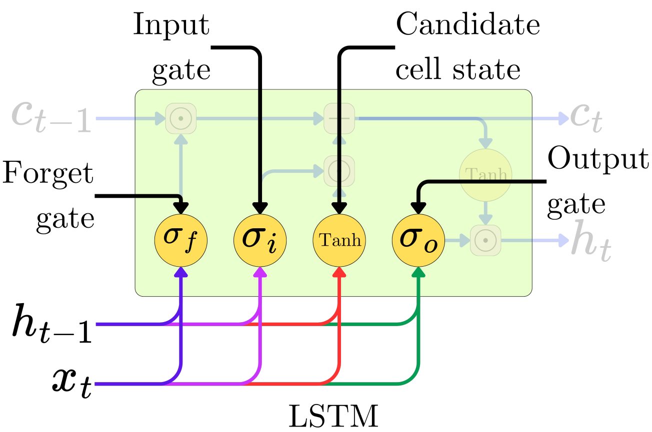
$$z\sigma(x) = \begin{cases} 0 & \text{if} \quad x \to -\infty \\ z & \text{if} \quad x \to +\infty \end{cases}$$



Furthermore, we mix the information of $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ through a fourth layer generating a candidate cell state:

$$\tilde{\mathbf{c}}_t = \tanh\left(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1} + \mathbf{b}_c\right) \tag{5}$$

This time, we use the $\tanh$ function that ranges from -1 to $+1$.



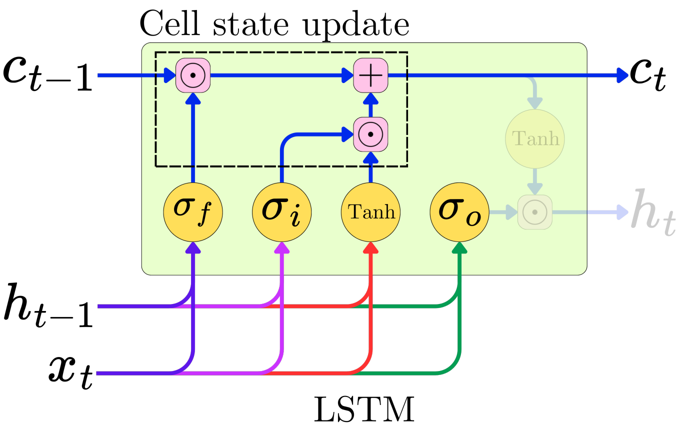
We call $\tilde{\mathbf{c}}_t$ the new candidate cell state, and $\mathbf{c}_{t-1}$ is the cell state of the previous iteration. We are going to use the forget gate $\mathbf{f}_t$ to decide how much of the old state we should keep or forget and the input gate $\mathbf{i}_t$ to decide how much of the new candidate we should add:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{6}$$

$\mathbf{c}_t$ is the current cell state and $\odot$ denotes the elementwise multiplication.

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \odot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ a_3 b_3 \end{bmatrix}$$

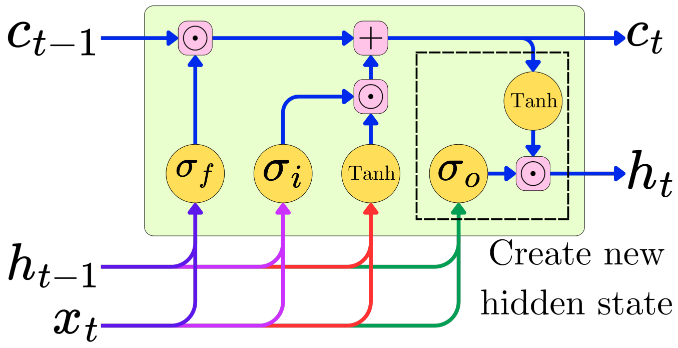
The term $\mathbf{f}_t \odot \mathbf{c}_{t-1}$ means "keep" (or forget) some fraction of the old cell state, and $\mathbf{i}_t \odot \tilde{\mathbf{c}}_t$ means "add" some fraction of the candidate state.



Finally, we compute which part of the cell state gets exposed as the new hidden state $\mathbf{h}_t$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{7}$$

The output gate $\mathbf{o}_t$ determines how much of the cell state $\mathbf{c}_t$ to transform and send out as $\mathbf{h}_t$.



LSTM

2 Long-Term Memory VS Short-Term Memory

In LSTM, we see that the information is carried from one iteration to the next by the cell state $\mathbf{c}_t$ and the hidden state $\mathbf{h}_t$. The dependency of a cell state $\mathbf{c}_t$ on the cell state $\mathbf{c}_{t-1}$ of the previous iteration is captured by the update equation:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

If we compute the gradients in one dimension, we get:

$$\frac{\partial c_t}{\partial c_{t-1}} = f_t \tag{8}$$

Therefore, for T time steps, the cell states form a dependency chain:

$$\mathbf{c}_1 \to \mathbf{c}_2 \to \cdots \to \mathbf{c}_T \tag{9}$$

and the gradients accumulate over the chain. Formally, in one dimension:

$$\frac{\partial c_T}{\partial c_1} \approx \prod_{t=2}^{T} f_t \tag{10}$$

We start the product at $t = 2$ because, in most practical implementations of LSTM, both the previous cell state $\mathbf{c}_{t-1}$ and the previous hidden state $\mathbf{h}_{t-1}$ at the very first time step are simply initialized to zero. We know that $0 < f_t < 1$ because f_t is the result of a logistic function. Therefore, no mechanism makes it possible for the gradients to explode. If the training data contains text samples with long-term information worth remembering, the network can learn W_f, U_f, and $\mathbf{b}_f$ parameters such that some elements of $\mathbf{f}_t$ remain close to 1. If each $f_t \approx 1$, the product stays near 1, and there is no exponential decay. That is why $\mathbf{c}_t$ is well-suited for long-term memory. This effect is typically known as the "constant error carousel," where the backpropagation of the errors can decay slowly over many steps. However, we do have the constraint $f_t < 1$, and even if the forget gate is 0.99, over 100 steps $0.99^{100} \approx 0.366$. That's still a noticeable decay, but it is much slower than if it was 0.5, for example, which would decay exponentially faster.

On the other hand, the hidden state $\mathbf{h}_t$ is dependent on the previous hidden state through:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Because $\mathbf{h}_t$ depends on $\mathbf{o}_t$ and $\tanh \mathbf{c}_t$, it can change more quickly from step to step than $\mathbf{c}_t$. Even if $\mathbf{c}_t$ remains stable, the output gate $\mathbf{o}_t$ might vary, and the product $\mathbf{o}_t \odot \tanh(\mathbf{c}_t)$ can fluctuate notably from step to step, reflecting more immediate or "short-range" changes. If we consider the gradients from one step to the next:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} \times \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_{t-1}} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t} \times \frac{\partial \mathbf{c}_t}{\partial \mathbf{h}_{t-1}} \tag{11}$$

The output gate derivative $\frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t}$ can be small or large depending on the sigmoid slope. The derivative of $\tanh(\mathbf{c}_t)$ is $1 - \tanh^2(\mathbf{c}_t)$ which is near 0 if $\mathbf{c}_t$ saturates. The partial derivative $\frac{\partial \mathbf{c}_t}{\partial \mathbf{h}_{t-1}}$ itself branches into the forget/input gates and the candidate cell state, each of which can saturate or diminish signals. This means that the gradient route from $\mathbf{h}_T$ to $\mathbf{h}_1$ accumulates multiple gating factors bounded within $[0, 1]$. Over many steps, it is easy for them to decay. You can also get partial blow-up if the gates push in certain directions strongly, but long-range signals are more likely to degrade when traveling through $\mathbf{h}_1$. This is why $\mathbf{h}_t$ is better suited for short-term memory.

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.